На этом занятии мы с вами познакомимся с одной из ключевых возможностей ООП языка C++ - **виртуальными методами классов**. Во многом именно виртуальные методы совместно с наследованием переводят нас на абстрактный уровень проектирования программного кода без необходимости сразу прописывать какую-либо конкретику. Также **виртуальные методы – это яркий представитель настоящего динамического полиморфизма**. В отличие от статического, с которым мы сталкивались при перегрузке функций и вызовов обычных методов базовых классов через объекты дочерних классов.

Начнем с того, что у нас имеются два уже знакомых нам класса: базовый GeomBase для обобщенного представления геометрических фигур на плоскости; Line – класс для описания объектов прямых линий:

```cpp
class GeomBase {
protected:
    int x0{0}, y0{0}, x1{0}, y1{0};
public:
    void set_coords(int x0, int y0, int x1, int y1)
    {
        this->x0 = x0; this->y0 = y0;
        this->x1 = x1; this->y1 = y1;
    }
};

class Line : public GeomBase {
private:
    double length{0.0};
public:
    void draw() const
    {
        printf("Line: %d, %d, %d, %d\n", x0, y0, x1, y1);
    }
};
```

Я их записал в упрощенном виде, чтобы ничто нас не отвлекало от магии виртуальных методов. Давайте посмотрим, что будет, если в классе GeomBase тоже объявить публичный метод draw с той же сигнатурой:

```cpp
class GeomBase {
protected:
    int x0{0}, y0{0}, x1{0}, y1{0};
```

```
public:
...
    void draw() const
    {
        printf("GeomBase: %d, %d, %d, %d\n", x0, y0, x1, y1);
    }
};
```

Теперь, если в функции main создать объект дочернего класса Line и вызвать через него метод draw:

```language-cpp
int main()
{
    Line* ptr_ln = new Line;

    ptr_ln->draw();

    delete ptr_ln;
    return 0;
}
```

то будет вызван метод дочернего класса Line и в консоли отобразится строка:

```
Line: 0, 0, 0, 0
```

Однако если привести указатель объекта Line к указателю типа GeomBase и через него вызвать метод draw:

```language-cpp
int main()
{
    Line* ptr_ln = new Line;
    GeomBase* ptr_b = ptr_ln;

    ptr_ln->draw();
    ptr_b->draw();

    delete ptr_ln;
    return 0;
}
```

то вызовется уже метод базового класса:

```
Line: 0, 0, 0, 0
GeomBase: 0, 0, 0, 0
```

Об этом мы с вами подробно уже говорили. Если же в базовом классе не будет метода draw, то команда:

```cpp
ptr_b->draw();
```

приведет к ошибке: метод не найден.

Но, что если нам нужно сделать так, чтобы через указатель базового класса вызывался метод дочернего класса? Как раз для этого и предназначены виртуальные методы. Если в базовом классе метод draw обозначить, как виртуальный:

```cpp
class GeomBase {
...
public:
...
    virtual void draw() const
    {
        printf("GeomBase: %d, %d, %d, %d\n", x0, y0, x1, y1);
    }
};
```

то при выполнении программы в консоли увидим строчки:

```
Line: 0, 0, 0, 0
Line: 0, 0, 0, 0
```

Но, если в дочернем классе убрать метод draw и снова запустить программу, то увидим:

```
GeomBase: 0, 0, 0, 0
GeomBase: 0, 0, 0, 0
```

Метод был взят из базового класса, так как в дочернем он отсутствует.

Вернем метод draw в дочернем классе Line и вместо операции приведения типа указателя создадим объект базового класса:

```cpp
int main()
{
    Line* ptr_ln = new Line;
    GeomBase* ptr_b = new GeomBase;

    ptr_ln->draw();
    ptr_b->draw();
```

```
    delete ptr_ln;
    delete ptr_b;
    return 0;
}
```

Теперь в консоли появляются строки:

```
Line: 0, 0, 0, 0
GeomBase: 0, 0, 0, 0
```

То есть, самостоятельный объект базового класса ведет себя так, словно у него нет никаких дочерних классов. И это логично, мы же создаем именно объект класса GeomBase и именно с ним, как с единым целым собираемся работать, поэтому никаких переопределений виртуальных методов здесь быть не должно.

Класс, который переопределяет или наследует виртуальный метод, еще называются **полиморфным** (polymorphic class).

В нашем примере класс Line является полиморфным.

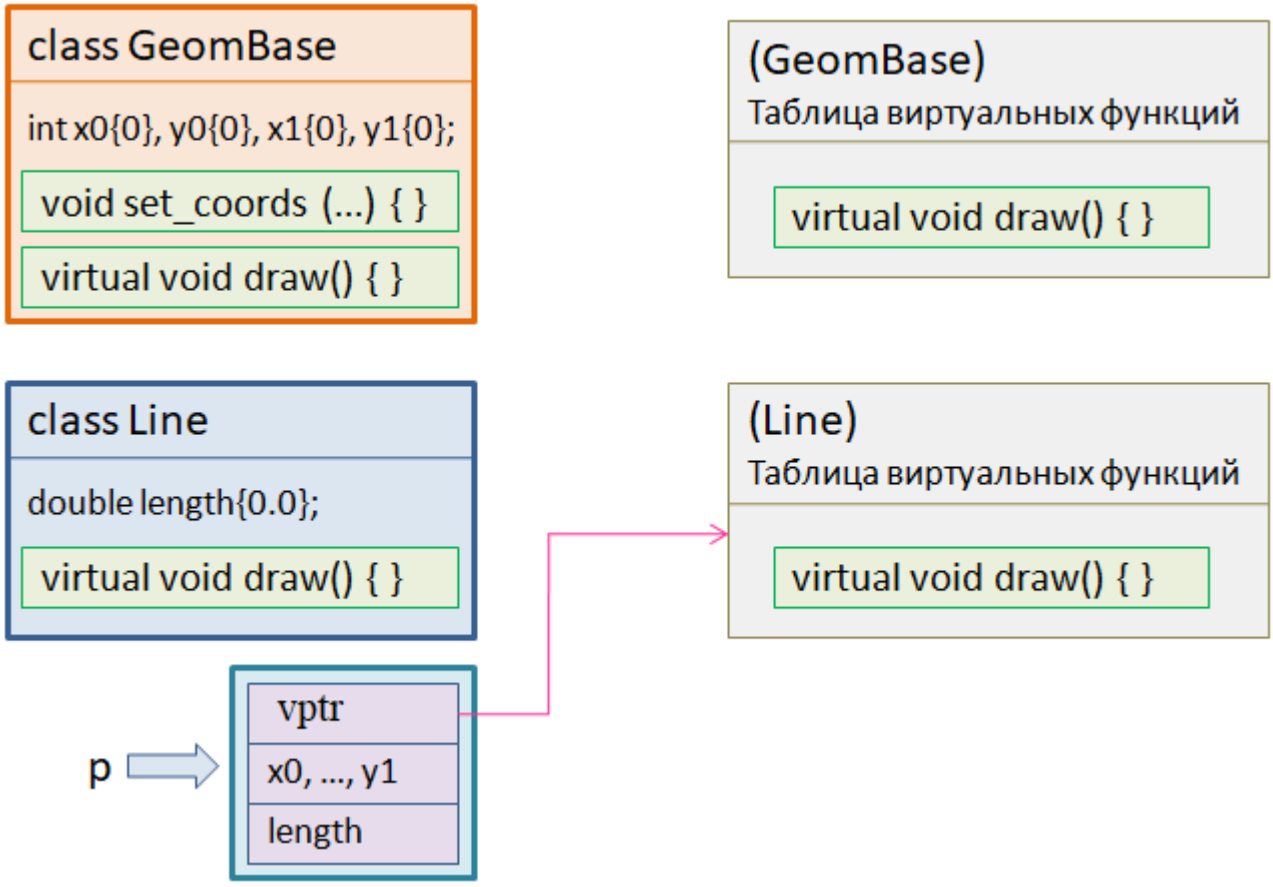
Как работают виртуальные методы

Это была демонстрация объявления и работы виртуального метода. Давайте теперь детальнее изучим принцип, поднаготную их работы.

Когда создается объект класса, в котором имеется хотя бы одна виртуальная функция (например, класса Line):

```language-cpp
Line* p = new Line;
```

то компилятор автоматически в этом объекте размещает специальный указатель vptr на таблицу виртуальных функций, связанной с этим классом:



В нашем примере таблица для класса Line будет состоять из одной виртуальной функции (метода) draw. **Обратите внимание, что, несмотря на то, что метод draw в дочернем классе Line не помечен как виртуальный, тем не менее, компилятором он рассматривается как виртуальный**. Поэтому часто у таких методов при их переопределении прописывают ключевое слово virtual, чтобы подчеркнуть факт наличия виртуального метода.

Итак, при создании объекта p вначале помещается указатель vptr на таблицу виртуальных функций класса Line. Зачем нужна эта таблица? Во-первых, теперь при вызове метода draw:

```language-cpp
p->draw();
```

компилятор читает значение указателя vptr, по нему переходит к таблице виртуальных методов (функций), находит там нужный виртуальный метод и формирует код его вызова.

Во-вторых, если выполнить операцию приведения типа к указателю на базовый класс:

```language-cpp
GeomBase* b = p;
```

А, затем, через него вызвать метод draw:

```cpp
b->draw();
```

то **компилятор обратится все к тому же объекту p, прочитает значение его указателя vptr и вызовет метод draw для класса Line, а не базового класса GeomBase, как это было бы без наличия таблицы виртуальных функций**.

Если же мы создаем изначально объект базового класса GeomBase:

```cpp
GeomBase* b = GeomBase;
```

то указатель vptr будет вести на таблицу виртуальных функций именно этого класса и вызов:

```cpp
b->draw();
```

будет связан с методом draw класса GeomBase.

Вот так, через таблицу виртуальных методов реализуется механика их вызовов.